Implicit Deep Learning Solver Report

Release 0.9

Erik Boee, Guillaume Goujard, Marius Landsverk & Axel Roenold

Dec 10, 2019

CONTENTS

CHAPTER

ONE

INTRODUCTION

Introduction to the report

The Implicit Deep Learning (IDL) Python package was designed as project for UC Berkeley's EE227BT. The theory and the training algorithm were devised in [1] and [2]. The purpose of the report is to explain precisely the training algorithm that we implemented and to present some of the first results that we could obtain. Finally, we will discuss the limits of our training algorithm and we will propose some improvements that can be implemented based on our feedback.

Introduction to the code

The python package is available at https://github.com/GuillaumeGoujard/IDLEstimator. It was developed to implement two main methods : fit and predict. Though the package has been implemented for regression purpose, small changes can be added to provide the user with a classifier. We will show in section 6 how to effectively build your IDL model.

1.1 Implicit Deep Learning

Given an input $u \in \mathbb{R}^n$, where n denotes the number of features, we define the implicit deep learning prediction rule $\hat{y}(u) \in \mathbb{R}^n$ with ReLU activation

$$\begin{aligned} \hat{y}(u) &= Ax + Bu + c \\ x &= (Dx + Eu + f)_+, \end{aligned} \tag{1.1}$$

where $(.)_+ := \max(0, .)$ is ReLU activation, $x \in \mathbb{R}^h$ is called the hidden variable (h is the number of hidden features), $\Theta := (A, B, c, D, E, f)$ are matrices and vectors of appropriate size, they define the parameters of the model. The hidden variable x is implicit in the sense that there is in general no analytical formula for it, this is different from classical deep learning for which, given the model parameters, the hidden variables can be computed explicitly via propagation through the network.

1.2 Notation and definitions

We denote by $\|.\|$ the euclidean norm, $\|.\|_2$ the corresponding norm (i.e. the spectral norm) and $\|.\|_F$ the Frobenius norm. $\mathbb{R}^n_+$ denotes the positive orthant of the vector space $\mathbb{R}^n, \mathbb{S}^n$ the set of real symmetric matrices of size n and $\mathbb{S}^n_+$ the cone of positive semi-definite matrices of size n. The transpose of a matrix or vector is denoted $.^T$ and

elementwise product is denoted $\odot$. Given a differentiable function f from $\mathbb{R}^{n\times p}$ to $\mathbb{R}$ we define the scalar by matrix partial derivative in denominator layout convention as

$$\frac{\partial f}{\partial A} = \nabla_A f = \begin{bmatrix} \frac{\partial f}{\partial A_{1,1}} & \cdots & \frac{\partial f}{\partial A_{1,p}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial A_{n,1}} & \cdots & \frac{\partial f}{\partial A_{n,p}} \end{bmatrix} \in \mathbb{R}^{n\times p}.$$

We say that a function $(x, y) \to f(x, y)$ with separable domain of definition $\mathcal{X} \times \mathcal{Y}$ is bi-convex in (x, y), if for all $x \in \mathcal{X}$, the function $y \to f(x, y)$ is convex and for all $y \in \mathcal{Y}$ the function $x \to f(x, y)$ is convex. We say that a function is smooth if it is differentiable and its gradient is Lipschitz continuous. We say that f is bi-smooth if it is smooth in x given y and vice-versa. An example of bi-smooth and bi-convex function is $(x, A) \to x^T Ax, A \in \mathbb{S}^n_+$.

1.3 Well-posedness

We say that matrix D is well-posed for (1.1) if there exists a unique solution $x = (Dx + \delta)_+ \; \forall\, \delta \in \mathbb{R}^h$. Using the fact that ReLU is 1-Lipschitz we have for $x_1, x_2 \in \mathbb{R}^h$

$$\|(Dx_1 + \delta)_+ - (Dx_2 + \delta)_+\| \leq \|D(x_1 - x_2)\| \leq \|D\|_2 \|x_1 - x_2\|.$$

If $\|D\|_2 < 1$ we have that the map $x \to (Dx + \delta)_+$ is a strict contraction. In that case, Banach's contraction mapping theorem applies, showing that the equation $x = (Dx + \delta)_+$ has a unique solution. In that case, a solution x can be computed via the Picard iterations

$$x^{k+1} = (Dx + \delta)_+, k = 1, 2, \cdots .$$

Note that $\|D\|_2 < 1$ is only a sufficient condition for well-posedness. Nevertheless this is the only condition we will consider in this article.

Note on the code In our implementation, we chose $\|D\|_2 \leq \frac{1}{2}$ to boost the Picard iterations.

CHAPTER

TWO

FORMULATION FOR IDL

See the bibliography for in-depth explanation.

2.1 Problem formulation

Let us consider the input and output data matrices $U = [u_1, \cdots, u_m] \in \mathbb{R}^{n\times m}, Y = [y_1, \cdots, y_m] \in \mathbb{R}^{p\times m}$ with m being the number of datapoints - the corresponding optimization regression problem (i.e. with squared error loss) reads

$$\begin{aligned} \min_{X,\Theta} \quad & \mathcal{L}(Y, [\Theta, X]) := \frac{1}{2m}\|AX + BU + c1_m^T - Y\|_F^2 \\ \text{s.t.} \quad & X = (DX + EU + f1_m^T)_+ \\ & \|D\|_2 < 1, \end{aligned} \tag{2.1}$$

where 1_m is a column vector of size m consisting of ones. For clarity we have highlighted in blue the optimization variables. The non-convexity of this problem arises from the nonlinear implicit constraint and the matrix product terms AX and DX. In practice we replace the constraint $\|D\|_2 < 1$ by the closed convex form $\|D\|_2 \leq 1 - \epsilon$, where $\epsilon > 0$ is small.

2.2 Fenchel Divergence Lagrangian Relaxation

Using Fenchel-Young inequality, it can be shown that the equation $x = (Dx + Eu + f)_+$ is equivalent to

$$\begin{cases} \mathcal{F}(x, Ax + Bu + c) = 0 \\ x \geq 0 \end{cases} \tag{2.2}$$

with the Fenchel Divergence $\mathcal{F}$ defined by

$$\mathcal{F}(x_1, x_2) := \frac{1}{2}x_1 \odot x_1 + \frac{1}{2}(x_2)_+ \odot (x_2)_+ - x_1 \odot x_2.$$

We use the term divergence because by construction $\mathcal{F}(x_1, x_2) \geq 0 \; \forall x_1, x_2 \in \mathbb{R}_+^h \times \mathbb{R}^h$. Given $X = [x_1, \cdots, x_m]$ and $Z = [z_1, \cdots, z_m]$ we write

$$\mathcal{F}(X, Z) := \frac{1}{m}\sum_{i=1}^{m} \mathcal{F}(x_i, z_i).$$

A Lagrangian relaxation approach to solving (2.1) problem using the implicit constraint formulation (2.2) consists in solving given a dual variable $\lambda \in \mathbb{R}_+^h$

$$\begin{aligned} \min_{X\geq 0,\Theta} \quad & \mathcal{L}(Y, [\Theta, X]) + \lambda^T \mathcal{F}(X, DX + EU + f1_m^T) \\ \text{s.t.} \quad & \|D\|_2 < 1, \end{aligned} \tag{2.3}$$

This problem is bi-smooth in $[\Theta, X]$, but it is not convex or bi-convex. Nevertheless we can make it bi-convex with extra conditions on Θ as shown in the next section.

2.3 Linear matrix inequality parameter constraints for bi-convexity

Let us define $\Lambda = \text{diag}(\lambda) \in \mathbb{S}_+^h$

Theorem 1. *Problem* (2.3) *is bi-convex in* $[\Theta, X]$ *if we impose one of the two following feasible linear matrix inequalities (LMI)*

$$\Lambda - (\Lambda D + D^T \Lambda) \in \mathbb{S}_+^h \tag{2.4}$$

$$\Lambda + A^T A - (\Lambda D + D^T \Lambda) \in \mathbb{S}_+^h \tag{2.5}$$

Proof. The loss term $\mathcal{L}(Y, [\Theta, X])$ is already bi-convex in (Θ, X), but it is not the case for the Fenchel Divergence term $\lambda^T \mathcal{F}(X, DX + EU + f1_m^T)$, which is not convex in X in the general case. A sufficient condition for this term to be convex in X given Θ is for the following function

$$x \to \lambda^T (\frac{1}{2} x \odot x - x \odot Dx) = \frac{1}{2} x^T (\Lambda - (\Lambda D + D^T \Lambda)) x,$$

to be convex. This term is convex in x if the LMI (2.4) is satisfied. Now the second LMI similarly arises by leveraging the fact that we can also use the term in the loss to make the objective convex in x. Indeed the objective function of (2.1) is convex in x if

$$x \to \frac{1}{2} x^T A^T A x + \frac{1}{2} x^T (\Lambda - (\Lambda D + D^T \Lambda)) x,$$

is convex, which corresponds to LMI (2.5). It might not be obvious that (2.5) is actually an LMI, but using Schur complement we can prove it is equivalent to

$$- \begin{bmatrix} I_p & A \\ A^T & \Lambda D + D^T \Lambda - \Lambda \end{bmatrix} \in \mathbb{S}_+^{p+h}.$$

If D satisfies (2.4) then it satisfies (2.5). We imediately have that $D = \delta I_n$ with $\delta \leq \frac{1}{2}$ satisfies (2.4) (and $\|D\|_2 \leq 1 - \epsilon$). Which proves that both LMIs are feasible.

2.4 Bi-convex Formulation

From this proof, the problem formulation reads

$$\begin{aligned} \min_{X \geq 0, \Theta} \quad & \mathcal{L}(Y, [\Theta, X]) + \lambda^T \mathcal{F}(X, DX + EU + f1_m^T) \\ \text{s.t.} \quad & \|D\|_2 \leq 1 - \epsilon \\ & \Lambda + A^T A - (\Lambda D + D^T \Lambda) \in \mathbb{S}_+^h, \end{aligned} \tag{2.6}$$

this problem is well-posed -feasible solutions exist- and bi-smooth.

Note on the code In our implementation, the LMI constraint combined with $||D||_2 < 1$ constraint turned out to be very computationally heavy. By using the alternative constraint $||D||_\infty < 1$ we obtained much quicker projections, without noticeable change in the predictions. However, this is not justified in theory.

CHAPTER

THREE

GRADIENT DESCENTS

3.1 Block coordinate descent and first order methods

As problem (2.6) is bi-convex, a natural strategy is the use of block coordinate descent (BCD): alternating optimization between Θ and X. BCD corresponds to the following algorithm,

for $k = 1, 2, \cdots$ **do**

$$\begin{aligned}\Theta^k \leftarrow \text{argmin}_{\Theta} & \frac{1}{2m}\|AX^{k-1} + BU + c1_m^T - Y\|_F^2 + \lambda^T \mathcal{F}(X^{k-1}, DX^{k-1} + EU + f1_m^T) \\ & \Lambda + A^T A - (\Lambda D + D^T \Lambda) \in \mathbb{S}_+^h, \\ & \|D\|_2 \leq 1 - \epsilon \\ X^k \leftarrow \text{argmin}_{X \geq 0} & \frac{1}{2m}\|A^k X + B^k U + c^k 1_m^T - Y\|_F^2 + \lambda^T \mathcal{F}(X, D^k X + E^k U + f^k 1_m^T)\end{aligned}$$

end

In practice such updates might be to heavy computationally as the number of datapoints m increase, or as the model size increases (i.e. h, n or p). Instead we propose to do block coordinate projected gradient updates. This method is also considered to be better at avoiding local minima. Let us denote

$$\mathcal{G}(\Theta, X) := \mathcal{L}(Y, [\Theta, X]) + \lambda^T \mathcal{F}(X, DX + EU + f1_m^T)$$

In the remainder of this section we derive the gradients $\nabla_\Theta \mathcal{G}(\Theta, X), \nabla_X \mathcal{G}(\Theta, X)$ and corresponding 'optimal' step-sizes using the Lipschitz coefficients of the gradients- which is the advantage of having a bi-smooth optimization problem. Note that the objective $\mathcal{G}$, given X is separable in $\Theta_1 := (A, B, c)$ and $\Theta_2 := (D, E, f)$. Using scalar by matrix calculus

$$\begin{cases} \nabla_A \mathcal{G}(\Theta, X) = \Omega(A, B, c)X^T \in \mathbb{R}^{p \times h} \\ \nabla_B \mathcal{G}(\Theta, X) = \Omega(A, B, c)U^T \in \mathbb{R}^{p \times n} \\ \nabla_c \mathcal{G}(\Theta, X) = \Omega(A, B, c)1_m \in \mathbb{R}^p \end{cases},$$

with $\Omega(A, B, c) := \frac{1}{m}(AX + BU + c1_m^T - Y) \in \mathbb{R}^{p \times m}$. Hence we can show that a Lipschitz constant for the gradient is given by

$$L_{\Theta_1}(X) := \frac{1}{m} \max(m, \|X\|_2^2, \|U\|_2^2, \|XU^T\|_2),$$

and the 'optimal' step-size for gradient descent is then simply given by

$$\alpha_{\Theta_1}(X) := \frac{1}{L_\Theta(X)}.$$

Regarding the gradient with respect to Θ_2, we have

$$\begin{cases} \nabla_D \mathcal{G}(\Theta, X) = \Omega(D, E, f, \Lambda) X^T \in \mathbb{R}^{h \times h} \\ \nabla_E \mathcal{G}(\Theta, X) = \Omega(D, E, f, \Lambda) U^T \in \mathbb{R}^{h \times n} \\ \nabla_f \mathcal{G}(\Theta, X) = \Omega(D, E, f, \Lambda) 1_m \in \mathbb{R}^h \end{cases},$$

with $\Omega(D, E, f, \Lambda) := \frac{\Lambda}{m}\left((DX + EU + f1_m^T)_+ - X\right) \in \mathbb{R}^{h \times m}$, we can show that a Lipschitz constant for the gradient is

$$L_{\Theta_2}(X) := \frac{\lambda_{\max}}{m} \max(m, \|X\|_2^2, \|U\|_2^2, \|X\|_2 \|U\|_2),$$

where $\lambda_{\max} = \max_{j \in \{1, \cdots, h\}} \lambda_j$. We can then similarly define an 'optimal' step-size $\alpha \Theta_2$. We have that

$$\nabla_X \mathcal{G}(\Theta, X) = \frac{1}{m}\left\{A^T(AX + BU + c1_m^T - Y) + (\Lambda - \Lambda D - D^T \Lambda)X + D^T \Lambda (DX + EU + f1_m^T)_+ - \Lambda(EU + f1_m^T)\right\}.$$

A Lipschitz constant for this gradient is

$$L_X(\Theta) = \frac{1}{m}(\|A^T A + \Lambda - \Lambda D + D^T \Lambda\|_2 + \lambda_{\max} \|D\|_2^2).$$

We can then take the step-size $\alpha_X(\Theta) = \frac{1}{L_X(\Theta)}$. We propose the following block coordinate projected gradient scheme (BC-gradient) to find a candidate solution to 2.6. We denote compactly the convex set

$$\mathcal{S}_\Theta := \{\Theta | \Lambda + A^T A - (\Lambda D + D^T \Lambda) \in \mathbb{S}_+^h, \|D\|_2 \leq 1 - \epsilon\}$$

and $\mathcal{P}_{\mathcal{S}_\Theta}$ the corresponding convex projection

for $k = 1, 2, \cdots$ **do**

$$\Theta^k = \mathcal{P}_{\mathcal{S}_\Theta}\left(\Theta^k - \alpha_\Theta(X^{k-1}) \nabla_\Theta \mathcal{G}(\Theta^{k-1}, X^{k-1})\right)$$
$$X^k = \left(X^{k-1} - \alpha_X(\Theta^k) \nabla_X \mathcal{G}(\Theta^k, X^{k-1})\right)$$

end

Note on the code In our implementation, $\mathcal{P}_{\mathcal{S}_\Theta}$ was computationally heavy for large dataset. We chose to relax the bi-convex constraints by not enforcing the LMI constraints any more. Furthermore, notice that $||D||_{+\infty} < 1$ also implies that the Picard iterations converges. Finally, instead of projecting on S_Θ, we project onto:

$$\mathcal{S}_D := \{D | \|D\|_\infty \leq \frac{1}{2}\}$$

CHAPTER

FOUR

DUAL ASCENTS

4.1 Dual methods

We propose the following schemes to find an appropriate dual variable λ. Let $\epsilon > 0$ be a precision parameter for the implicit constraint, i.e. such that we would have

$$\mathcal{F}(X, DX + EU + f1_m^T) \leq \epsilon$$

We start with $\lambda = 0$ and we solve the two following separate problems

$$\min_{X>0, A, B, c} \frac{1}{m} \|AX + BU + c1_m^T - Y\|_F^2$$

and then

$$\min_{D, E, f} 1_h^T \mathcal{F}(X, DX + EU + f1_m^T).$$

If $\mathcal{F}^* := \mathcal{F}(X, DX + EU + f1_m^T) < \epsilon I_h$ then we stop there. Otherwise, we proceed by the dual update method described below.

4.2 Dual ascent conditional on Fenchel Divergence

$$\lambda \leftarrow \lambda + \alpha \mathcal{F}^* \odot 1\{\mathcal{F}^* \geq \epsilon I_h\}, \tag{4.1}$$

where $\alpha > 0$ is a step-size. Note that here we only update the components of λ for which the corresponding Fenchel divergence is more than ϵ. We then proceed to solve (2.6) using previously discussed methods and iterate. Alternatively, if the BC-gradient method is used, we can do a dual update after each BC-gradient update. Both methods were implemented in our code.

CHAPTER

FIVE

PARAMETERS AND HYPERPARAMETERS

For the code, certain parameters are to be tuned by the user. The others have been fixed.

5.0.1 Given by the User

1. **dual learning rate** : α
2. **tolerance fenchel** : ϵ
3. **starting lambda** : $\lambda_0 =$ starting lambda $* 1_h$,
4. **rounds number** : number of time the dual variables are updated in the loop.
5. **inner tolerance** : to early stop the gradient descent with respect to X and θ.

5.0.2 Implementation of IDL

class IDL.**IDLModel**(*hidden_variables=1*, *dual_learning_rate=0.1*, *tol_fenchel=0.1*, *inner_tol=0.001*, *starting_lambda=None*, *initialization_theta=None*, *random_state=0*, *early_stopping=True*, *verbosity=True*)

Implementation of the Scikit-Learn API for Implicit Deep Learning

Parameters

- **hidden_variables** – int number of hidden variables of the vector X (see
- **dual_learning_rate** – float Positive float, Dual learning rate (IDL's "alpha")
- **tol_fenchel** – float Positive float, Fenchel tolerance threshold for dual's update (IDL's "alpha")
- **inner_tol** – float Positive float, tolerance threshold for early stopping in the gradient descents with respect to theta
- **verbosity** – bool Verbosity of the training process.
- **random_state** – int Random number seed for initialization.

:param initialization_theta

Full documentation of parameters can be found here: https://github.com/GuillaumeGoujard/IDLEstimator/blob/master/docs/source/sections/introduction.rst.

fit(*X*, *y*, *rounds_number=100*, *verbose=True*, *type_of_training='two_loops'*, *eval_set=None*)

Fit IDL Model

Parameters

- **X** – array_like Feature matrix
- **y** – array_like Labels
- **rounds_number** – int Maximum rounds number in the outer loop
- **verbose** – bool
- **type_of_training** – string Two types of training :
  - ”two_loops” : RECOMMENDED, we optimize theta and X variables and then we do one step of dual ascent.
  - ”one_loop” : one iteration is going to successively operate one step of gradient descent and one step

  of dual ascent

Returns self : object Returns self.

predict(*X*, *k_iterations=10000*)

Predicting function. :param X: array-like

The input sample.

Parameters **k_iterations** – int Maximum number of Picard iterations

Returns y: array-like Returns a prediction array.

CHAPTER

SIX

EXAMPLE : LINEAR REGRESSION

The purpose of this chapter is to show to the user how to effectively create a IDL Model.

6.1 Setup

The package is compatible with Python version 3 or higher only. The code is located at the following url https://github.com/GuillaumeGoujard/IDLEstimator

```
git clone + https://github.com/GuillaumeGoujard/IDLEstimator
```

The main class "IDLEstimator" is located in IDL/IDLtemplate/IDL.py.

```
import IDL as idl
```

Listing 6.1: Import the IDL module

6.2 Use case : Linear regression

6.2.1 Creating the inputs

We create randomly 1000 iid datapoints (X_i, ϵ_i) such that

$$X_i \sim U(0,1)$$

$$\epsilon_i \sim \mathcal{N}(0, 0.5)$$

We compute the output :

$$Y_i = X_i + \epsilon$$

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression

number_of_data_points = 1000
test_data_point = int(0.1*number_of_data_points)
plot_training = True

def create_regressive_model(n_samples, a=1, noise_std = 0.01):
```

```
    X = np.random.random_sample(n_samples)
    return model(X, a=a, noise_std=noise_std)

def model(X, a=2, noise_std =0.01):
    n_samples = X.shape[0]
    epsilons = np.random.normal(0, noise_std, n_samples)
    return X.reshape((-1, 1)), a * X + epsilons


X, y = create_regressive_model(number_of_data_points, a=1, noise_std=0.5)
X_train, y_train, X_test, y_test = X[:-test_data_point], y[:-test_data_point], X[-test_data_point
    :], y[-test_data_point:]
```

Listing 6.2: Create the linear model

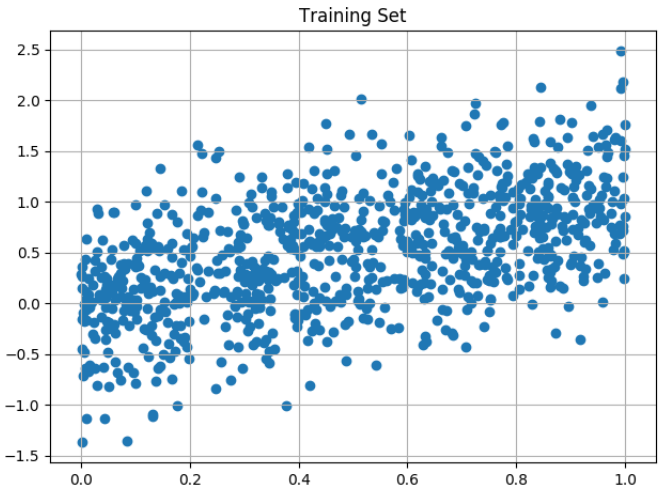

Fig. 1: Training dataset.

6.2.2 Creating and fitting an IDL Model

```
"""
IDL model
"""
IDL = idl.IDLModel(hidden_variables=1, dual_learning_rate=0.1, tol_fenchel=0.1, inner_tol=1e-3,
    starting_lambda = None, initialization_theta = None, random_state=0, early_stopping=True,
                verbosity=True)
IDL.fit(X_train, y_train, rounds_number=50, verbose=True, type_of_training="two_loops", eval_set
    = (X_test, y_test)
```

Listing 6.3: Create the linear model

6.2.3 Predicting

```
y_pred = IDL.predict(X_test)
```

Listing 6.4: Predict

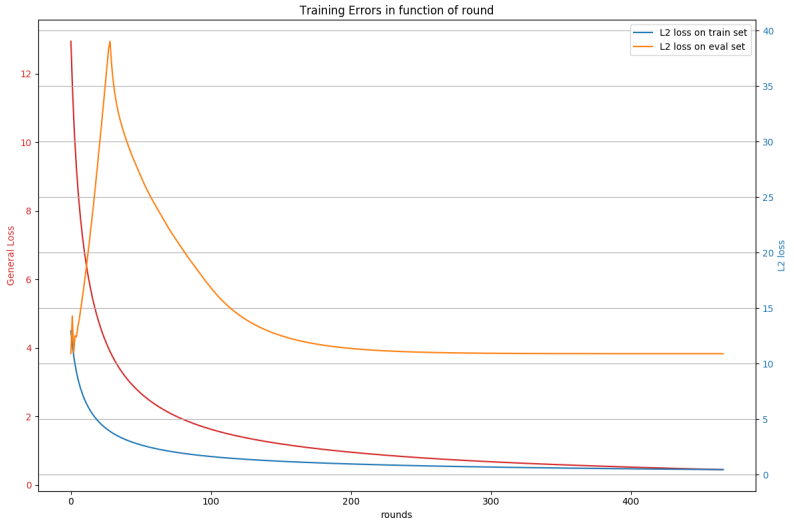

Fig. 2: Training curves : in red the drop in the general loss, in blue the L2 Loss over the train dataset and in yellow the L2 Loss over the eval dataset (the test set in this case) in function of the rounds

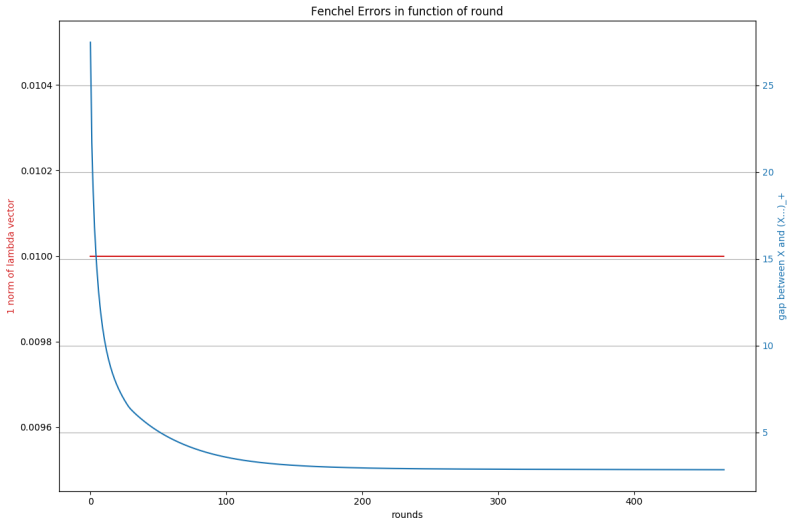

Fig. 3: L1 norm of the dual variables in red and $X^{k+1} - (DX^k + EU + f)_+$ in blue in function of the rounds

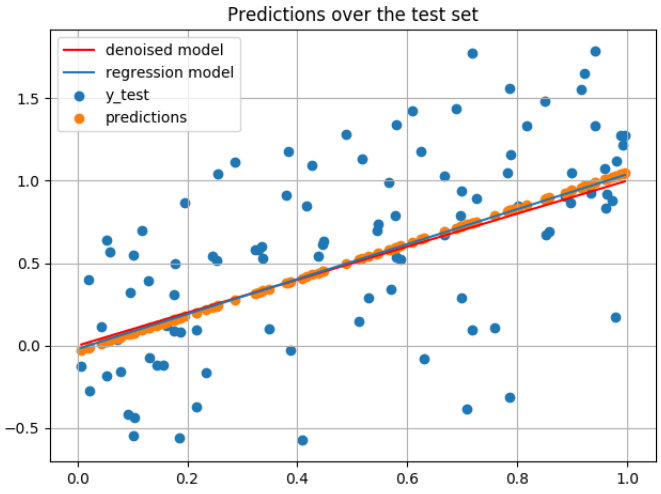

Fig. 4: Predictions over the test dataset

CHAPTER

SEVEN

EXAMPLES

7.1 Implicit Generative Process

We take 10 features, 10 000 samples, a 1-dimensional output, and 500 hidden variables. Then, we take A, B, c, D, E, f random and sparse. D is projected so that $||D||_\infty < 1$. U, X taken as random normal.

We solve with picard iterations :

$$X = (DX + EU + f1_m^T)_+$$

Then, we simulate

$$y = AX + BU + c1_m^T$$

The objective is to recover the model or at least to beat the linear regression on this specific example.

Results

After initializing and training the IDL model with $\alpha = 1e-5, \epsilon = 1e-5, h = 300$, we get an rmse of 0.0789 vs 0.262 for a linear regression.

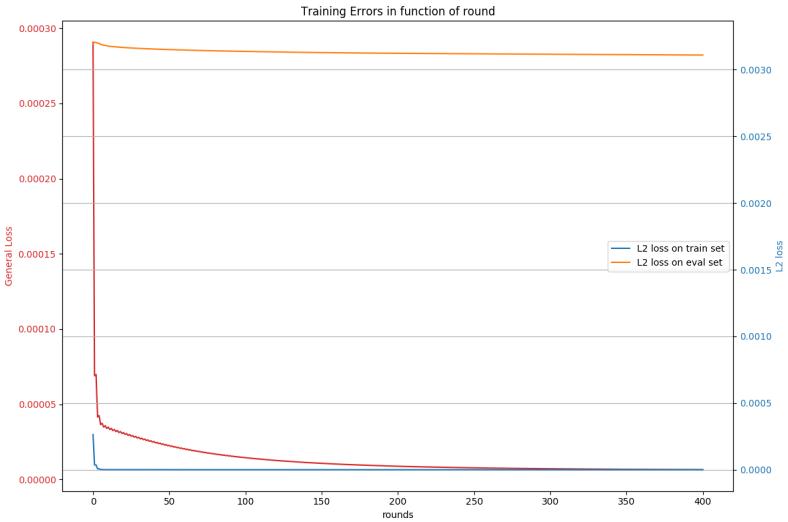

Fig. 1: The drop of general loss and of the L2 loss over the training set is not followed by the L2 Loss over the evaluation set. Strong suspicion of overfitting.

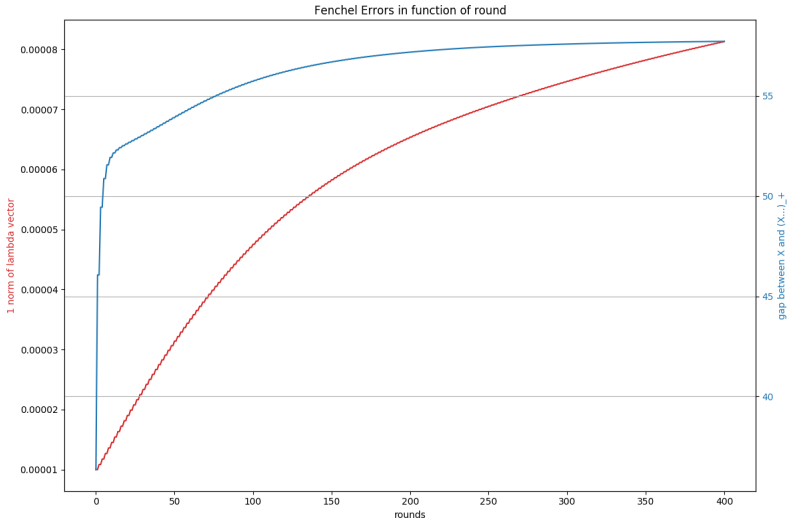

Fig. 2: If the dual variables are correctly updated, the dual learning rate seems too small to have a real impact on the fenchel error

7.2 Regression on non-linear data

In addition to the example with linear regression, we wanted to test the model on an example with a non-linear relationship between the input parameters and the output data. We generated a data set from a sinus curve on $[0, 2\pi]$ with noise generated from the normal distribution $N(0, 1)$. The results, i.e. predictions, on this dataset was much more dependent on the hyperparameters and the initial training. For a good initialization we were able to obtain quite good predictions, as presented in figure 3. However, when the initialization was bad, as presented in figure 4, it was much

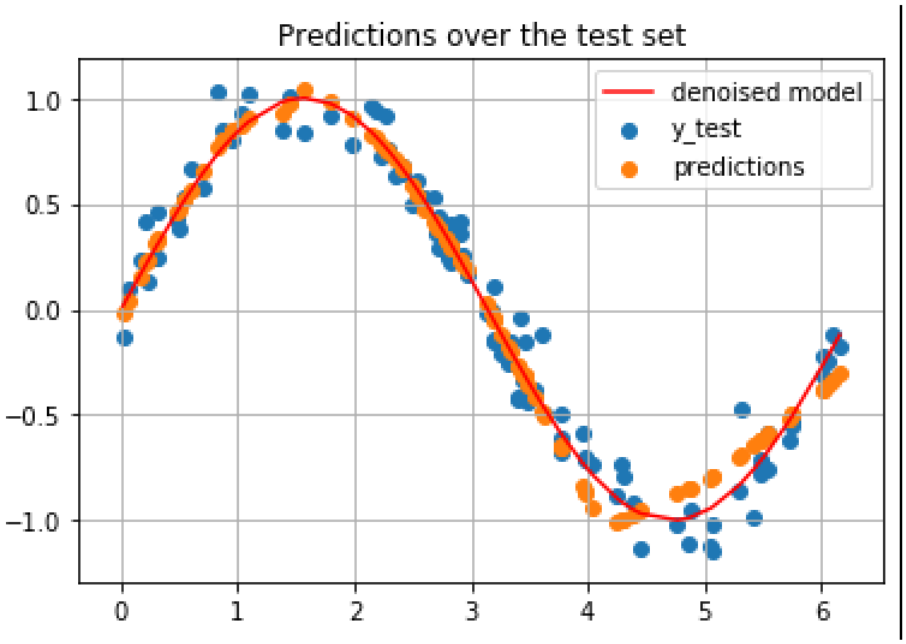

Fig. 3: Prediction over 100 test data points.

more difficult to train the model to give good predictions. For instance, when λ or the learning rate α was too large,

we observed the case displayed in figure 10, that the reduction of the Fenchel divergence is weighted much heavier than the reduction of the L2loss. Later we discovered that altering the initialization so that we first choose D, E, f at random, before choosing X as the unique solution of the implicit constraint, and further optimizing over A, B, c gave a very good initialization. However, at times when the initialization is good, we observe that there is little to none improvement in the general loss during training, it is clear that the optimization in D, E, f is not working well.

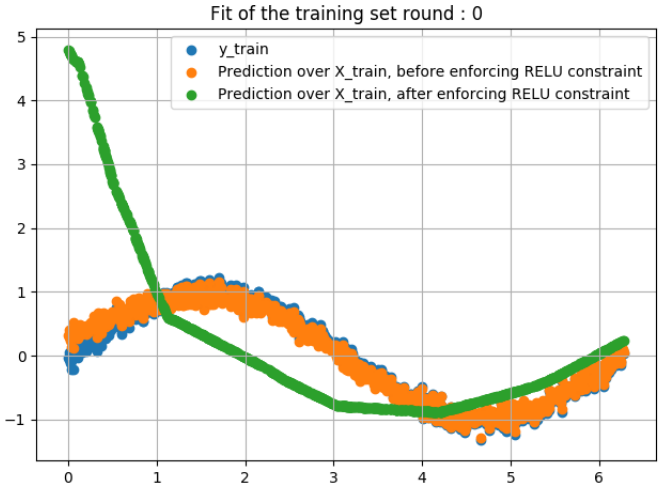

Fig. 4: Fit on the training set after initialization. The orange is the overfitted $AX + BU + c1_m^T$ before enforcing the RELU constraint, the green is the prediction after enforcing $X = (DX + EU + f)_+$.

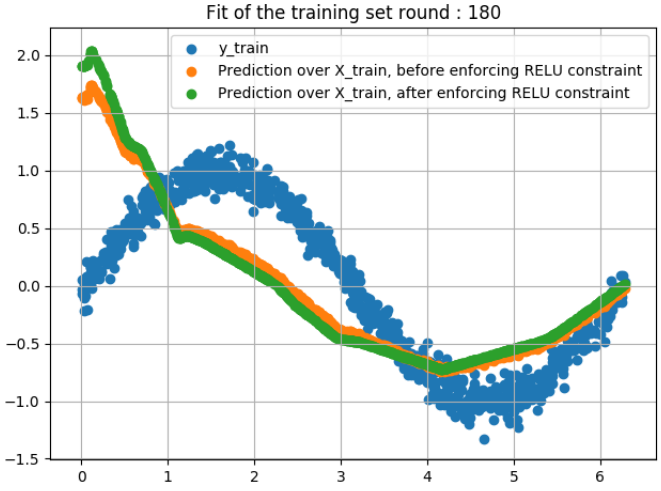

Fig. 5: Fit on the training set after 180 rounds of training. The orange is $AX + BU + c1_m^T$ before enforcing the RELU constraint, the green is the prediction after enforcing $X = (DX + EU + f)_+$.

7.3 Main Insights

We want to assess the respective impact of each of the main hyper-parameters on our last example. We recall that those hyper-parameters are $h, \alpha, \epsilon, \lambda_0$, resp, the number of hidden variables, the dual learning rate, the fenchel tolerance threshold and the starting value for the dual variables.
The experimental setup is the following : we initialize 10 (θ, X) using the function "initialize_theta" that we feed into an IDL model. Having each of the other factors fixed, we look at the influence of each parameter independently. Since we have 10 simulation we display the mean of the L2 error as a triangle and the standard deviation as a vertical line.

7.3.1 The training does not help

Our first and most crucial insight is that the essential work is done at the initialization. The following plot was realized after running the training algorithm with $\alpha = \epsilon = \lambda_0 = 10^{-6}$ and $h = 100$. After initializing (θ, X), we look at the L2 loss over the test set (x axis). After the end of the training, we look again at the L2 loss over the test set (y axis). Surprisingly, we make more errors after the training than before.

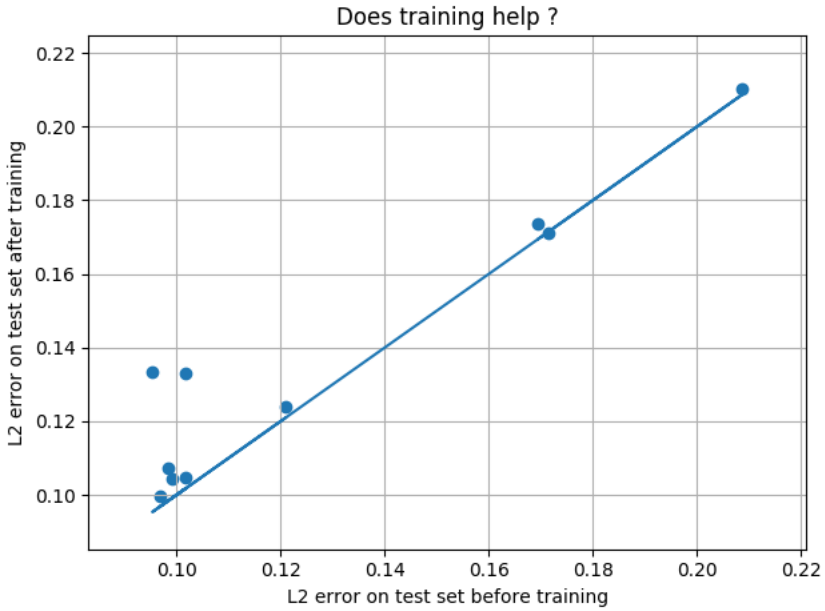

Fig. 6: Better off without training

7.3.2 Influence of hidden variables

The following plot was realized after running training with different numbers of random variables (x axis). The other parameters were $\alpha = \epsilon = \lambda_0 = 10^{-6}$. It shows that the more we use hidden variables, the more we can hope to capture the non linearity.

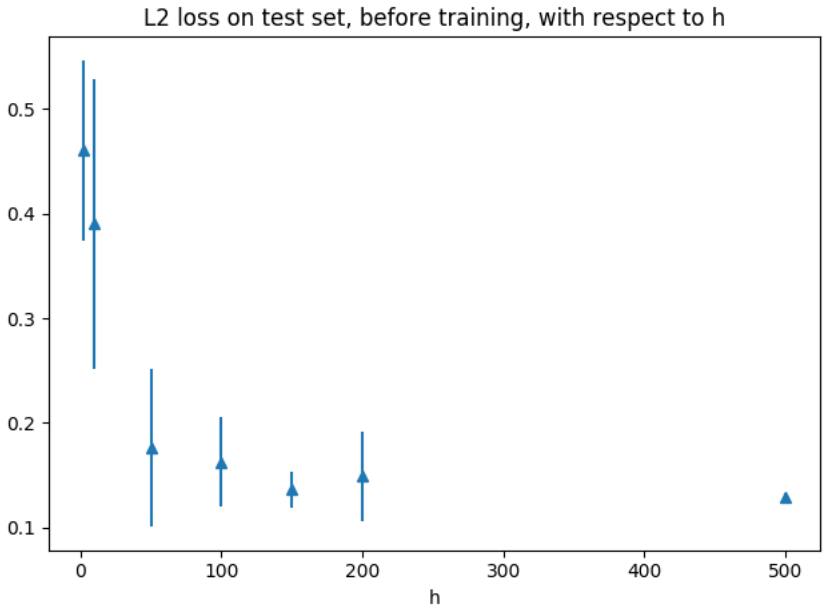

Fig. 7: The more hidden variables, the more accurate the model is

7.3.3 Influence of dual learning rate

The following plot was realized after running training with different dual learning rate (x axis). The other parameters were $\epsilon = \lambda_0 = 10^{-6}$ and $h = 200$. It shows that there are no significant impact of the dual learning rate on the accuracy of our model.

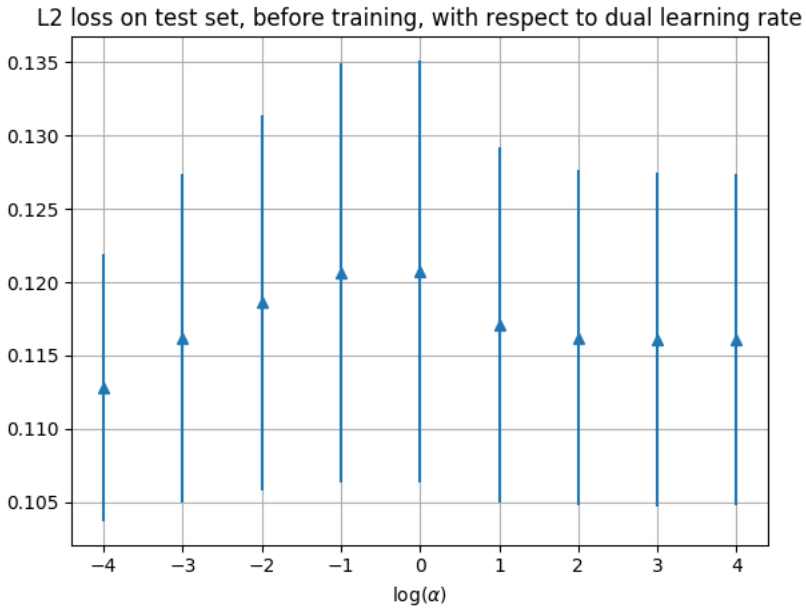

Fig. 8: No influence of the dual learning rate

7.3.4 Influence of the fenchel tolerance value

The following plot was realized after running training with different values for the fenchel tolerance (x axis). The other parameters were $\alpha = \lambda_0 = 10^{-6}$ and $h = 200$. It shows that there are no significant impact of the fenchel tolerance on the accuracy of our model.

Fig. 9: No influence of the Fenchel tolerance value

7.3.5 Influence of starting lambda

The following plot was realized after running training with different values for starting lambda (x axis). The other parameters were $\alpha = \epsilon = 10^{-6}$ and $h = 200$. In average, it shows that the greater starting lambda is the less accurate will be. We can nonetheless have some good surprises.

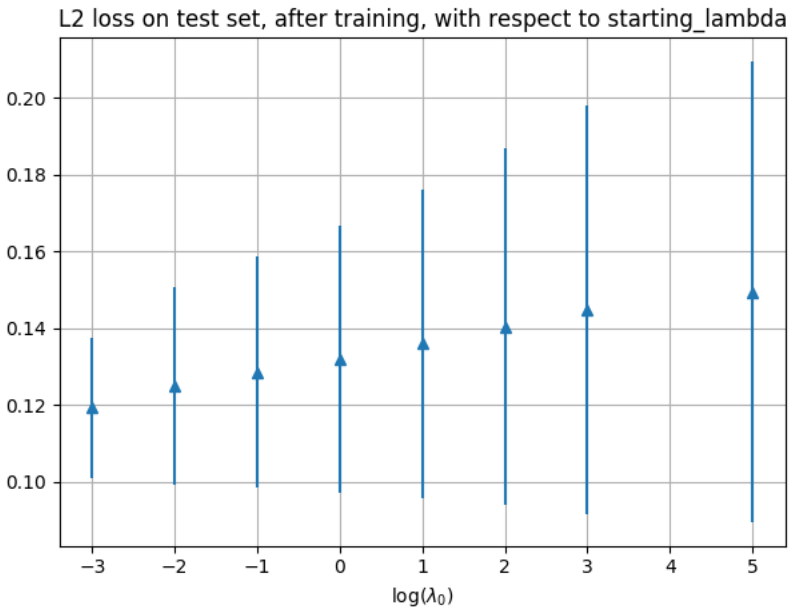

Fig. 10: No influence of starting lambda

CHAPTER

EIGHT

CONCLUSION - FURTHER WORK

To conclude, our implementation of an Implicit Deep Learning Solver package on python yields good yet limited results. An IDL model trained by our package, recovers easily linear relationships. It recovers also more precisely implicit generative process than a linear regression model would. However, the results are limited with nonlinear data. After the initialization, the training becomes more challenging, and no significant progress is made in training the values of D, E and f. This hurts the generalization of the model, hence its predictive capacity and its precision.

If we focused our attention into the effective programming of the training algorithm, we devoted our validation work on finding "good practices" for the initialization and tuning hyper-parameters. In this regard, we observed that initializing D, E, f at random, solving for X by Picard iteration and then optimizing over A, B, c yields a good initialization. However, the model rarely improves during the training phase and this is regardless of the hyperparameters that are selected, as shown in the results. One may investigate whether the choice of training algorithms and more specifically of the learning rate for D, E, f may be adapted to improve the performance during the training. There is also more work to be done to make our IDL solver more robust to different initializations. One may experiment quitting training once the L2 error on a held-out validation set is starting to increase again to limit the over-fitting effect that we witnessed in our examples.

In our work we have mainly investigated regression problems where both the input and output dimension are one-dimensional. Further investigations should involve assessing the accuracy of the IDL model on classification tasks with multiple covariates and evaluate if the implicit model behaves the same in this setting.

One may also experiment with different architectures for the IDL model, for instance by specifying the model to have a convolutional structure.

Finally, we wanted to thank Bertrand Travacca for introducing us to this fascinating notion of Implicit Deep Learning, and for helping us to overcome some problems that we faced concerning the theory underlining the training of such a model.

BIBLIOGRAPHY

[1] L. El Ghaoui, F. Gu, B. Travacca, A. Askari, *Implicit Deep Learning*,arXiv:1908.06315(2019)

[2] B. Travacca, *Bi-convex Implicit Deep Learning* (2019)